JAX-RS Implementations: A Performance Comparison

John Velandia, Sonia Rios, Holman Bolivar, Juan Vanzina, Nicolas Almanzar
Universidad Católica de Colombia, Colombia.
javelandia@ucatolica.edu.co

***Abstract*—Restful services are implemented around the world to integrate software systems. JAX-RS is a standard API proposed by Java to maintain a common architectural pattern independently of the provider´s implementation (libraries). At the moment, there is no study regarding when to use any of the implementations, thus, the aim of this article is to compare implementations considering different test scenarios that would help software architects and developers to make the right decision when performance variables are a selection criteria. This research carries out a methodology based on stability, peak, stress and load variables. Additionally, the software architecture is presented for some of the implementations studied to ensure that they are comparable.**

***Index Terms*—Apache CXF; JAX-RS; Jersey; Performance; REST; RESTeasy; RESTful Services; RESTlet.**

I. Introduction

The Representational State Transfer (REST) is an architectural style for distributed hypermedia systems [1]. It is based on principles that guaranties a common standard for exchanging data among information systems using client and server architecture [2]. REST uses as underlying protocol the Hypertext Transfer Protocol (HTTP) which offers standardized interfaces and implicit quality attributes such as interoperability and modifiability [2] as advantages. In addition, HTTP is a well-known protocol given that the World Wide Web is built based on this [3].

The growth of information systems with the need of interoperate with other information systems applies to any industry sector, for instance banking, e-commerce and social networks, (i.e., Facebook and Twitter). REST and SOAP technology are mostly used to cover this interoperability need. The use of REST has been rising because it is easy, simple and lightweight to build restful web services.

Java API for RESTful Web Services (JAX-RS) is a specification framework that defines how plain Java objects are bound to URIs and HTTP operations using Java annotations [4]. This framework is important since this establishes a standard way to handle incoming and outgoing server requests and information flows from one restful service to another; consequently, JAX-RS facilitates and simplifies a restful service implementation.

Providers have been implementing JAX-RS, supporting the REST principles: Addressability, uniform interface, content representation, stateless interaction and hypermedia. In addition, quality attributes such as security, thread-save, concurrency and performance are offered by providers. However, there has not been any research regarding which implementation is better in terms of these quality attributes. Considering that there is a wide range of quality attributes, and each of them is composed of metrics and methodologies to evaluate them, the objective of this paper is to assess the performance of the following implementations: Jersey, Resteasy, Restlet and CXF, because according to [5]-[10]they are the most used for integrating information systems.

A. *Statement of the problem*

Restful services are used equally in the industry and academic around the world [11], and software architects and developers always come up with the same question: Which JAX-RS implementation shall we use in terms of performance?

This question is solved as workaround by searching on web sites that lack of accuracy and reliability since there is not a deep assessment regarding JAX-RS implementations. A proof is that digital libraries do not provide studies about this, i.e. ACM Digital Library, Science Direct, Latin Index, Web of Science, IEEE Xplore, among others.

B. *Main contributions*

This research would allow organizations, namely software architects and developers, to make a choice based on the performance that the implementation presented in this paper has.

A new methodology is proposed for comparing the performance of JAX-RS implementations; thus, it may extrapolate to another scenario from which software systems need a formal comparison.

An architectural analysis of JAX-RS implementations is presented to understand which components are involved in the communication and what their main tasks are.

II. Methodology

The methodology comprises 8 activities. The first activity consists of the analysis of the following JAX-RS architectures: Jersey, RESTlet, RESTEasy and Apache CXF; this serves as the input of the following activity. The second activity involves defining software components that are due to be assessed. The third activity focuses on defining the quality attributes, in this case, the performance attributes. The fourth activity identifies metrics and variables included in the performance test. The aim of the following activity is to plan and design the test. The sixth activity prepares the environment to run the tests. The last two activities address tests repetitively and result analyses. Figure 1 summaries the exposed methodology.

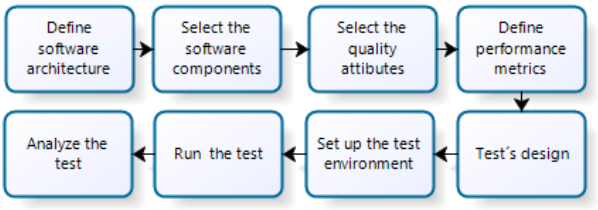

Figure 1: Activities of the proposed methodology

A. *Software architecture*

a. Jersey

This implementation has been developed by Oracle, and its aim is to support JAX-RS specification [12]. Despite Jersey implementation is widely used, there is not any formal and well-defined software architecture in papers, books and Jersey's documentation, for instance [5], [12], [13] and [14]. Consequently, the proposed architecture is based on the Oracle's documentation and Jersey's dependencies [15]

Core component is the backbone of this implementation; it is used for both server and client. Server component provides the necessary functionality to handle incoming and outgoing request, and also to deploy itself on HTTP servers [10]. JSR311 API is in charge of compiling restful server and client, since it defines the restful services API. Servlet component listens URIs request to bind them to resources and services. JSON component supports format representation requests [10], [12].

b. RESTlet

This a lightweight and comprehensive framework that implements JAX-RS (Sandoval 2009). It is considered as simple and scalable; it is designed for high concurrency (Restlet 2016). It supports both client and server by means of restful libraries. It also provides the following libraries as extensions to support Web standards: HTTP, SMTP, XML, JSON, OData, OAuth, RDF, RSS, WADL, and Atom (Louvel et al. 2012).

Security Restlet is based on HTTP features: authentication authorization, confidentiality and access login - reducing the needs to integrate and learn third party APIs, in this way productivity increases (Louvel et al. 2012).

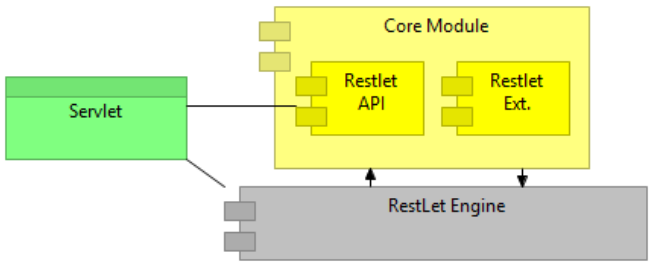

Figure 2: Restlet software architecture

The architecture encompasses a Core module that contains two components: (1) Restlet API which supports the concepts of REST and HTTP, handles server and client requests, and (2) Restlet Extensions that supports integration to other plugins or APIs. In addition to the Core module, the Restlet engine acts as the backbone of Restlet. [16] [17]. Figure 2 depicts the architecture and the components that make up Restlet.

c. RESTEasy

RESTEasy is not only a RESTful implementation, but also a JBoss's umbrella project that provides additional libraries to build RESTful web services [5]. It supports JAX-RS which means that restful principles are covered.

As in Jersey implementation, there is no formal architecture defined for RestEasy, and based on thi,s the architecture proposed is based on JBOSS´ documentation [18]. Servlet component listens incoming and outgoing server requests. Core component is the backbone of this library, since it supports restful features. Jaxb-provider is in charge of converting java objects into XML elements and vice versa. Multipart provider component is responsible for dealing with multiple formats, such as JSON, XML and others.

d. Apache CXF

CXF acronym comes from two projects, Celtix and XFire. Celtix is an open source Java-based Enterprise Service Bus (ESB) project. XFire, a Java-based SOAP framework, is an open source project from Codehaus [19][20]. CXF is an open source framework that supports JAX-RS implementation for building and developing Web Services. The aim of CFX is to simplify web services development.

This framework supports Java Script Object Notation (JSON) and XML data formats. It also provides notations to convert POJOs into restful Web Services. Additionally, it provides a set of tools to generate web service clients and web services based on standards, such as JAX-WS, WSDL, and SOAP [19][20]. Given the wide range of implementations, CXF is also well-known as a framework.

The architecture is composed of seven main components, as shown in Figure 3. Bus component is the backbone of CXF architecture, and it is in charge of providing a common application context for endpoint and shared objects. The advantage of having this common context is that it is used as a communication channel among components. A servlet is deployed to initialize the bus [20].

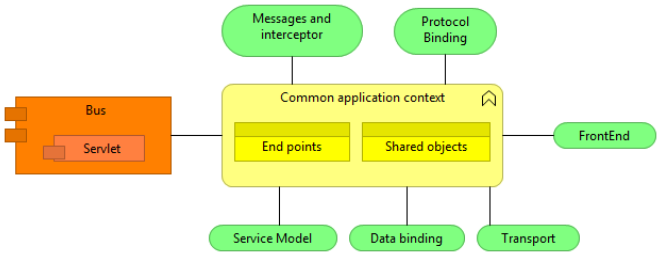

Figure 3: Apache CXF software architecture

The frontend component is responsible of creating web services using implementations such as JAX-RS and JAX-WS. Messaging and interceptors are components that head off incoming, outgoing and error messages exchanged between web service clients and server components. Service model component creates web services descriptions throughout Web Service Description Language (WSDL) artefact. Data Binding component maps and converts Java objects into XML elements and vice versa. Protocol binding maps and converts logical messages into physical data format that depends on the required protocol specific format. Transports components regards network details, i.e., the routing protocol, for instance HTTP or JMS [19].

B. Selection of software components to test

The aim of this activity is to ensure that components to be compared are comparable. Thus, the architectures of the four JAX-RS implementations are comparable, because they have the same objective, which is to support restful services. In details, the fourth architectures have three common components: (1) a servlet that receives requests, (2) an engine that processes requests and (3) a REST API provided by Java. In conclusion, the fourth implementations are equally comparable.

C. Quality attributes

The objective of this activity is to define the quality attributes that are due to use in the assessment. Since the aim of the research project is to measure the performance of the JAX-RS implementations, the quality attribute defined is performance.

D. Definition of Metrics

Metrics can be constructed to assess a variety of concerns, e.g., system or component technical performance, human-computer interaction, and process improvement. Using the top-down approach advocated in this framework, the metric selection is scoped by its parent EO. Likewise, each metric scopes and is informed by its associated measures.

The establishment of criteria and performance metrics are defined by the attributes of software quality within which are contemplated [5]:

a. Performance

It refers to the response time, use and performance of the system behavior. The following are the variables:

- *Time*: Total time the test lasts.
- *Requests*: The number of requests send to the server.
- *Completed* requests: number of requests per second that were completed in each test.
- *Dismiss*: number of requests cancelled in the test.
- *Failures*: number of requests that had failed.
- *Maximum value*: maximum number of requests.
- *Minimum:* minimum number of applications.
- *STD DEV*: standard deviation which measures the dispersion of values with respect to the average.

b. Efficiency

Quantity of resources and code required by a program or service to perform its function

c. Reliability

Degree in which a program is expected to perform its function with required accuracy.

E. Planning and designing the test

The objective of performance testing is to determine if the programmer is satisfied with the efficiency of implementation of the Framework of JAX-RS, under conditions of expected usage. There are four types of performance tests:

a. Load test

This type of testing is performed to observe the behavior of a service under defined number of requests. The load in our case considers the number of users that make requests to each Framework JAX-RS. For the implementation of the evidence, an initial charge of 100 requests per second is laid down. It gradually increases until it reaches the maximum load of requests per second, depending on the behavior's implementation. This test allows identifying possible bottlenecks and response times.

b. Stress test

The stress tests are intended to evaluate the behavior of the service at the time the requests are sent continuously, establishing if there are faults in memory. These sorts of tests are used to find the volume of data and the time software systems start to fail or are unable to respond to requests. In conclusion, this test leads a software system beyond the edge of normal circumstances.

c. Stability testing

Stability tests carry out a high number of requests to ensure the software system is still available; it looks for the limit of request that the system supports. The test consists of on leave implementation running over a time, registering if failures occur.

d. Peak tests

This test shows the behavior of the system by varying the number of requests dramatically to evidence the existence of anomalies in the violent change of requests per second. For example, the execution of the test sets an initial charge of 500 requests per second, which changes drastically to 12000 requests per second in a 5-minute period.

F. Test environment

This activity consists of setting the environment that would be used to run tests over the JAX_RS implementations. This activity encompasses software hardware, data structure and scenarios used for running tests.

a. Assumptions and restrictions

To ensure the performance test is accurate, the following assumptions and restrictions are considered: the implemented restful services are developed and deployed on the same server; communication network is not considered, because this variable could vary from time to time and it depends on the companies´ infrastructure. Thus, client and server are placed on the same server, which means that requests and responses are measured without network variables, i.e. latency and jitter; Data structure, length and weight of HTTP messages have the same content.

JAX-RS implementations are tested using the following sort of tests: (1) *Load test* to measure the number of transactions each library handles per second, (2) *Stability test* finds the limit of transactions per second that libraries support, (3) *Stress test* evidences the libraries´ behaviors in terms of performance under certain number of requests and (4) *peak tests* consists of sending blocks of request varying the number of them.

b. Tiers and layers architecture

In order to evaluate the JAX-RS implementations, four restful services are created, one for each implementation. The whole services are deployed on the same hardware server to guarantee the same variables. One tier is configured to run tests: one tier for the server and the client. As for the software layers, the prototype architecture comprises two layers: the service layer and the business model.

c. Hardware and software features

The server´s features are: processor accelerated AMD Quad-Core A6-5200 of 2.0 GHz; Microprocessor cache: 2MB cache; Memory: 4 GB DDR3 SDRAM with a maximum supported memory: 8 GB. Hard disk: 500GB drive (5400 RPM).

On software used: Operating system: Windows 8.1; LoadUI 1.0.1.; Apache CXF 3.1.2.; Jersey 2.21.; Restlet 2.3.4; RESTeasy 3.0.12.

d. Data structure

XML and JSON are used to build up data structures in order to determine the effectiveness of each one at the time of implementing them in the test scenarios.

Figure 4 and Figure 5 show the XML and JSON formats that are used in performance tests. These data structures correspond to the basic information of a person, which is stored for the scenario that uses database and only keep it in memory for the scenario that does not.

```
<?xml version="1.0" encoding="UTF-8"?>
<Personas>

  <Nombre>Nicolas</Nombre>
  <Apellido>Almanzar</Apellido>
  <Email>nalmanzar09@ucatolica.edu.co</Email>
  <Telefono>6485984</Telefono>
  <Edad>22</Edad>

</Personas>
```

Figure 4: Data structure in XML format

```
{
- Personas:[
        -{
                Nombre: "Nicolas",
                Apellido: "Almanzar",
                Email: "nalmanzar09@ucatolica.edu.co"
                Telefono: "6485984",
                Edad: "22"
         }
        ]
}
```

Figure 5: Data structure in XML JSON format

e. Scenarios

Figure 6 describes the first proposed scenario which has a MySQL database engine using JPA for the connection with the database, since JPA provides efficiency in connection and does not generate additional delays for the performance of each JAX-RS implementation.

This stage appears to estimate or calculate the times in that one incurs when there is a connection to a database, because the applications done by the developers include deals with relational databases.

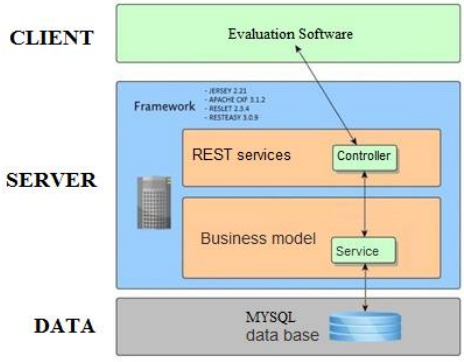

Figure 6: Test with a database architecture

Figure 7 describes the second testing scenario where the client does a number of requests to the JAX-RS Frameworks using XML and JSON, in this way formats in each Framework according to established performance tests to evaluate.

This scenario lacks of database engine, to avoid possible additional time.

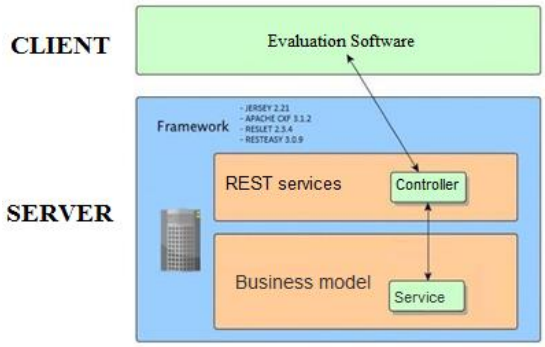

Figure 7: Test without a database architecture

f. The test and its results

Section III details out the discussion and obtained results.

III. Comparison by Type of Performance Test Framework

During the running test activity, it was noted that scenarios involving the database engine does not allow transparency for doing an adequate analysis, because the restful service requires more time to bring data from the database, even if JPA uses memory context. For this reason, the analysis must be carried out only with the results of the scenario that does not support database engine.

The results encompass the following metrics:

- *Time:* It is the time the test lacks.
- *Request:* Number of requests executed.
- *Completed request:* Number of completed request by the implementation.
- *Dismisses:* Number of requests dismissed.
- *Failures:* Number of failed requests.
- *Maximum value:* It is the maximum requests per second send to the implementation.
- *Minimum:* It is the minimum requests per second sent to the implementation.
- *Standard deviation (STD-DEV):* It is the standard deviation of the total requests. It is aggregated by using a weighted average.

Table 1 shows results of each implementation in the load test using JSON as the format of representation. The implementation that is capable of handling more requests per second is CXF, because during a period of 10 minutes, it reaches a value of 407.161 request/per second, with a standard deviation of 346 request/per second. While, Jersey is the less stable processing requests, which is evidenced by a standard deviation of 2155.

Table 1
Comparison of implementations in the load test

Metrics	Load			
	Jersey	Restlet	RESTeasy	CXF
	json	json	json	json
Time	10 Minutes	10 Minutes	10 Minutes	10 Minutes
Request	293991	305536	385172	407161
Completed Request	293989	305536	385172	407161
Dismisses	0	0	0	0
Failures	2	0	0	0
Maximum Value	213205	26802	16472	7627
Minimun Value	3	3	3	3
STD-DEV	2155,97	1329.44	949.95	346.67

Table 2 shows the best results of each Framework in stability test, which concludes that the CXF implementation XML format is efficient, because it performs as many requests for seconds in a 10-minute time period. The total number of failed requests is 0, requests discarded 0 requests the maximum value of requests is 300.003, with a maximum value of 7.598, this means that it responds to requests efficiently against other implementations.

Table 2
Comparison of implementations in the stability test

Metrics	Stability			
	Jersey	Restlet	RESTeasy	CXF
	xml	json	json	xml
Time	10 Minutes	10 Minutes	10 Minutes	10 Minutes
Request	300021	299928	300000	300003
Completed Request	30021	299928	300000	300003
Dismisses	0	0	0	0
Failures	0	0	0	0
Maximum Value	15012	24084	23265	7598
Minimun Value	3	4	3	3
STD-DEV	658,19	1588.79	1243.59	453.8

Like previous results, the best performance in terms of responses, Restlet outstands over other implementations when an XML format is required, because the total number of failed requests is 0, requests discarded 0 requests the maximum value of requests is 271.973, with a maximum value of 30656, as presented in Table 3.

Table 3
Comparison of implementations in the stress test

Metrics	Stress			
	Jersey	Restlet	RESTeasy	CXF
	xml	xml	json	json
Time	5 Minutes	5 Minutes	5 Minutes	5 Minutes
Request	215888	271973	47956	631614
Completed Request	215888	271973	47956	631609
Dismisses	0	0	2386	631614
Failures	0	0	0	5
Maximum Value	22328	30656	473783	55432
Minimun Value	5	5	25	3
STD-DEV	1209,79	797.29	11440.86	1396.11

Table 4 presents that in the test of peaks, the better performance is obtained from the CXF implementation, which answers a greater number of requests, the total number of failed requests is 0, requests discarded 0 requests the maximum value of requests is 335.517, with a maximum value of 22241.

Table 4
Comparison of implementations in the Test of Peaks

Metrics	Peaks			
	Jersey	Restlet	RESTeasy	CXF
	json	xml	json	xml
Time	5 Minutes	5 Minutes	5 Minutes	5 Minutes
Request	135672	135773	101684	335517
Completed Request	135460	135773	101419	335517
Dismisses	0	0	0	0
Failures	212	0	265	0
Maximum Value	445704	43792	32890	22241
Minimun Value	4	16	88	3
STD-DEV	11371,14	2709.12	2826.66	883.57

Then the general implementations according to the number of completed requests were evaluated successfully, so determined a scale from 0 to 10 where a score is set by each performance tests, to subsequently obtain a weighted value as shown in Table 5 and in this way compares the behavior of each one of the implementations.

Table 5
Ratings of the implementations

	Load	Stability	Stress	Peaks	Weighted Value
Jersey	4	2	9	4	4,75
Restlet	6	8.6	10	5	5,25
RESTeasy	8	9.7	3	5	4
Apache CXF	10	10	4	10	8,5

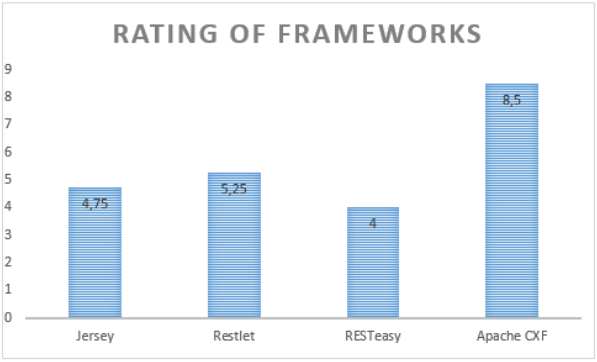

Figure 8: Bar Chart Rating of Frameworks

The behavior of all evaluated implementations is similar, however, the Apache CXF Framework shows superiority over others according to the established qualification, Figure 8.

IV. Future Work

Since the Jersey´s documentation does not provide its software architecture, it would be fruitful to research on its software components and their relations. Additionally, a deep dive among these implementations in terms of software architecture would help the academy and industry to develop new strategies regarding performance.

A provider method may be called multiple times at once. Therefore, it is important for the provider methods to be thread-safe. Lastly, the provider instance is relieved and destroyed by the garbage collector. Some of the implementations do not say anything about this item.

V. CONCLUSIONS

Regarding load test, the CXF is the fastest processing implementation, despite it is the less able to process requests per second. Jersey is the most capable to process requests per second, with some failures request, while RestLet and REST easy have similar behavior in processing request.

Interoperability between software systems using an efficient restful implementation would ensure a great performance as long as the chosen implementation matches particular needs, such as data format, simultaneous requests among others.

Performance is not the only quality attribute and the unique decision factor to choose a JAX-RS implementation, it is just one criteria of selection.

The implementations that are easier to implement are Jersey and Restlet, because the amount of lines of code is less. It was evidenced during the coding phase of the restful services.

According to the obtained results, one could conclude that as long as the server processes short messages, the performance improves. Additionally, if the software system is saturated, the response time of individual responses is affected negatively.

Independent of the implementation, one could conclude that JSON format performs better than XML format, because the length of the message is lighter; this is evidenced when a thousand of requests were executed by the server. In summary, data transfer using JSON is faster than XML.

In cases that a load test scenario is applied, and it requires a great performance, it is suggested to use CXF with JSON format. If a stability test is needed, and it requires a great performance, it is convenient to use CXF and XML formats. If, on the contrary, a stress test needs to be run, Restlet with XML format is indicated to implement the service. And if a scenario with peaks appears, the best option in terms of performance would be CXF with XML format.

In general, Apache CXF implementation would be the best choice for most of the scenarios.

REFERENCES

[1] R. O. Y. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," vol. 2, no. 2, pp. 115–150, 2002.

[2] B. Costa, P. F. Pires, F. C. Delicato, and P. Merson, "Evaluating REST architectures — Approach , tooling and guidelines," *J. Syst. Softw.*, vol. 112, pp. 156–180, 2016.

[3] S. Schreier, "Modeling RESTful applications," in *Proceedings of the Second International Workshop on RESTful Design - WS-REST '11*, 2011, p. 15.

[4] B. Burke, *RESTful Java with JAX-RS 2.0, 2nd Edition*. o'reilly, 2013.

[5] J. Sandoval, *RESTful Java Web Services: Master Core REST Concepts and Create RESTful Web Services in Java*. Packt Publishing Limited, 2009.

[6] C. Davis, "What if the Web Were Not RESTful?," in *Proceedings of the Third International Workshop on RESTful Design*, 2012, no. April, pp. 3–10.

[7] J. Strauch and S. Schreier, "RESTify : From RPCs to RESTful HTTP Design," pp. 11–18, 2012.

[8] X. Wu and H. Zhu, "Formalization and analysis of the REST architecture from the process algebra perspective," *Future Generation Computer Systems*, vol. 56, pp. 153–168, 2015.

[9] N. Balani and R. Hathi, *Apache CXF Web Service Development*. 2009.

[10] M. Hadley, S. Pericas-Geertsen, and P. Sandoz, "Exploring hypermedia support in Jersey," *Proc. First Int. Work. RESTful Des. - WS-REST '10*, p. 10, 2010.

[11] C. Pautasso and E. Wilde, "RESTful web services: principles, patterns, emerging technologies," in *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010, p. 1359.

[12] Oracle, "Jersey," *RESTful Web Services in Java*, 2017. [Online]. Available: https://jersey.java.net/. [Accessed: 13-Mar-2017].

[13] B. Burke, *RESTful Java with JAX-RS 2.0 - Designing and Developing Distributed Web Services*. O'Reilly Media, 2013.

[14] Oracle, "Types of Web Services," 2014. [Online]. Available: https://docs.oracle.com/javaee/7/tutorial/webservices-intro002.htm. [Accessed: 13-Mar-2017].

[15] Oracle, "Java Embedded Suite Application Developer's Guide: Working with Jersey," 2016. [Online]. Available: https://jersey.java.net/documentation/latest/jaxrs-resources.html. [Accessed: 13-Mar-2017].

[16] J. Louvel, T. Templier, and T. Boileau, *Developing RESTful web APIs in Java*. Manning Publications, 2012.

[17] Restlet, "Restlet user guide," *Restlet Framework*, 2017. [Online]. Available: https://restlet.com/open-source/documentation/user-guide/2.3. [Accessed: 13-Mar-2017].

[18] JBOSS Comunity, "RestEasy," *RESTEasy*, 2017. [Online]. Available: http://resteasy.jboss.org/. [Accessed: 13-Mar-2017].

[19] B. Naveen and H. Rajeev, *Apache CXF Web Service Development*. Birmingham, UK: Packt Publishing Ltd, 2009.

[20] A. S. Foundation, "Apache CXF Software Architecture Guide," *The apache software foundation*. [Online]. Available: http://cxf.apache.org/docs/cxf-architecture.html. [Accessed: 13-Mar-2017].